Contents

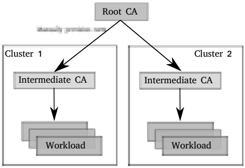
Plug in CA Certificates

4 minute read ✓ page test

This task shows how administrators can configure the Istio certificate authority (CA) with a root certificate, signing certificate and key.

By default the Istio CA generates a self-signed root certificate and key and uses them to sign the workload certificates. To protect the root CA key, you should use a root CA which runs on a secure machine offline, and use the root CA to issue intermediate certificates to the Istio CAs that run in each cluster. An Istio CA can sign workload certificates using the administrator-specified certificate and key, and distribute an administrator-specified root certificate to the workloads as the root of trust.

The following graph demonstrates the recommended CA hierarchy in a mesh containing two clusters.



CA Hierarchy

This task demonstrates how to generate and plug in the certificates and key for the Istio CA. These steps can be repeated to provision certificates and keys for Istio CAs running in each cluster.

Plug in certificates and key into the cluster

The following instructions are for demo purposes only. For a production cluster setup, it is highly recommended to use a production-ready CA, such as Hashicorp Vault. It is a good practice to manage the root CA on an offline machine with strong security protection.

1. In the top-level directory of the Istio installation package, create a directory to hold certificates and keys:

   ```
   $ mkdir -p certs
   $ pushd certs
   ```

2. Generate the root certificate and key:

   ```
   $ make -f ../tools/certs/Makefile.selfsigned.mk
    root-ca
   ```

   This will generate the following files:

   - `root-cert.pem`: the generated root certificate
   - `root-key.pem`: the generated root key
   - `root-ca.conf`: the configuration for

    `openssl` to generate the root certificate

    - `root-cert.csr`: the generated CSR for the root certificate

3. For each cluster, generate an intermediate certificate and key for the Istio CA. The following is an example for `cluster1`:

    ```
    $ make -f ../tools/certs/Makefile.selfsigned.mk
     cluster1-cacerts
    ```

    This will generate the following files in a directory named `cluster1`:

    - `ca-cert.pem`: the generated intermediate certificates
    - `ca-key.pem`: the generated intermediate key
    - `cert-chain.pem`: the generated certificate chain which is used by istiod

- `root-cert.pem`: the root certificate

  You can replace `cluster1` with a string of your choosing. For example, with the argument `cluster2-cacerts`, you can create certificates and key in a directory called `cluster2`.

  If you are doing this on an offline machine, copy the generated directory to a machine with access to the clusters.

4. In each cluster, create a secret `cacerts` including all the input files `ca-cert.pem`, `ca-key.pem`, `root-cert.pem` and `cert-chain.pem`. For example, for `cluster1`:

```
$ kubectl create namespace istio-system
$ kubectl create secret generic cacerts -n isti
o-system \
      --from-file=cluster1/ca-cert.pem \
      --from-file=cluster1/ca-key.pem \
      --from-file=cluster1/root-cert.pem \
      --from-file=cluster1/cert-chain.pem
```

5. Return to the top-level directory of the Istio installation:

   ```
   $ popd
   ```

Deploy Istio

1. Deploy Istio using the `demo` profile.

   Istio’s CA will read certificates and key from the secret-mount files.

   ```
   $ istioctl install --set profile=demo
   ```

Deploying example services

1. Deploy the `httpbin` and `sleep` sample services.

   ```
   $ kubectl create ns foo
   $ kubectl apply -f <(istioctl kube-inject -f sa
   mples/httpbin/httpbin.yaml) -n foo
   $ kubectl apply -f <(istioctl kube-inject -f sa
   mples/sleep/sleep.yaml) -n foo
   ```

2. Deploy a policy for workloads in the `foo` namespace to only accept mutual TLS traffic.

   ```
   $ kubectl apply -n foo -f - <<EOF
   apiVersion: security.istio.io/v1beta1
   kind: PeerAuthentication
   metadata:
     name: "default"
   spec:
     mtls:
       mode: STRICT
   EOF
   ```

Verifying the

certificates

In this section, we verify that workload certificates are signed by the certificates that we plugged into the CA. This requires you have `openssl` installed on your machine.

1. Sleep 20 seconds for the mTLS policy to take effect before retrieving the certificate chain of `httpbin`. As the CA certificate used in this example is self-signed, the `verify error:num=19:self signed certificate in certificate chain` error returned by the openssl command is expected.

   ```
   $ sleep 20; kubectl exec "$(kubectl get pod -l app=sleep -n foo -o jsonpath={.items..metadata.name})" -c istio-proxy -n foo -- openssl s_client -showcerts -connect httpbin.foo:8000 > httpbin-proxy-cert.txt
   ```

2. Parse the certificates on the certificate chain.

   ```
   $ sed -n '/-----BEGIN CERTIFICATE-----/{:start
   /-----END CERTIFICATE-----/!{N;b start};/.*/p}'
    httpbin-proxy-cert.txt > certs.pem
   $ awk 'BEGIN {counter=0;} /BEGIN CERT/{counter+
   +} { print > "proxy-cert-" counter ".pem"}' < c
   erts.pem
   ```

3. Verify the root certificate is the same as the one specified by the administrator:

   ```
   $ openssl x509 -in certs/cluster1/root-cert.pem
    -text -noout > /tmp/root-cert.crt.txt
   $ openssl x509 -in ./proxy-cert-3.pem -text -no
   out > /tmp/pod-root-cert.crt.txt
   $ diff -s /tmp/root-cert.crt.txt /tmp/pod-root-
   cert.crt.txt
   Files /tmp/root-cert.crt.txt and /tmp/pod-root-
   cert.crt.txt are identical
   ```

4. Verify the CA certificate is the same as the one specified by the administrator:

```
$ openssl x509 -in certs/cluster1/ca-cert.pem -text -noout > /tmp/ca-cert.crt.txt
$ openssl x509 -in ./proxy-cert-2.pem -text -noout > /tmp/pod-cert-chain-ca.crt.txt
$ diff -s /tmp/ca-cert.crt.txt /tmp/pod-cert-chain-ca.crt.txt
Files /tmp/ca-cert.crt.txt and /tmp/pod-cert-chain-ca.crt.txt are identical
```

5. Verify the certificate chain from the root certificate to the workload certificate:

```
$ openssl verify -CAfile <(cat certs/cluster1/ca-cert.pem certs/cluster1/root-cert.pem) ./proxy-cert-1.pem
./proxy-cert-1.pem: OK
```

Cleanup

- Remove the certificates, keys, and intermediate files from your local disk:

```
$ rm -rf certs
```

- Remove the secret `cacerts`, and the `foo` and `istio-system` namespaces:

```
$ kubectl delete secret cacerts -n istio-system
$ kubectl delete ns foo istio-system
```

- To remove the Istio components: follow the uninstall instructions to remove.